SAP Exchange Infrastructure
Mapping Functionality in XI
Overview Integration Builder

Mapping – Introduction
- Overview
- Java Mapping
- XSLT Mapping
- ABAP Mapping

Message Mapping
- General Concepts
- Standard Functions
- Element Cardinality
- Context Handling
- User-Defined Functions
- Value Mapping
- Examples
Overview Integration Builder
Mapping – Introduction
Message Mapping
Integration Builder

Common client-server framework for all objects

Two instances of the framework

■ Repository
  ◆ Business Scenarios & Business Processes
  ◆ Interface Objects
  ◆ Mapping Objects
  ◆ Adapter Objects

■ Directory
  ◆ Business Scenarios
    ● Receiver determination
  ◆ Interface determination
  ◆ Endpoints
  ◆ Logon Data
Objects in the Integration Repository

- **Business Scenario Objects**
  - Business Scenarios
  - Actions

- **Interface Objects**
  - Message Interfaces
  - Message Types
  - Fault Message Types
  - Data Types
  - Imported Objects (IDocs, RFCs)

- **Mapping Objects**
  - Interface Mappings
  - Message Mappings
  - Imported Archives

Unified behaviour of all objects for change management, naming conventions, documentation,...
Integration Builder: Introduction

The image shows a screenshot of the SAP Integration Builder with a focus on message mapping. The interface includes a tabbed view with sections for objects, change lists, and conflicts. The main area displays the design and text messages for message types, such as Contact and Customer, with details like fields, types, occurrences, and descriptions. The screen highlights data mapping, with examples of fields such as Name, Address, and City, along with data types like varchar and string. The interface also showcases the ability to manage business scenarios and build software components.
Object Handling in the Integration Repository

Before development can start

- Development manager or administrator has to import a software component version from SLD
- Create namespaces in this software component version

Each repository object is uniquely identified by the triple software component version, namespace, object name

As a general rule:

- Software component : Namespace = 1 : n
- A namespace is transferred to a new version of a software component after development is finished (Release-Transfer)

Object version management

- Changes are collected in (user-specific) change lists and can be activated once
- Each user may have different change lists for each software component version he is working in
- After activation a new object version is released. Old versions can be viewed by means of an object history.
## Integration Builder XI 2.0 SP1: Software Logistics

<table>
<thead>
<tr>
<th>Feature</th>
<th>Purpose</th>
<th>Objects from source -&gt; destination</th>
<th>‘Role’ of target object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>Reuse (global data types, for example)</td>
<td>Within one Repository: SWCV1-&gt;SWCV2, SWCV1-&gt;SWCV1, Within one Directory</td>
<td>Copy with initial object version</td>
</tr>
<tr>
<td>(context menu)</td>
<td></td>
<td>------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Release transfer</td>
<td>Take over (all) objects to next (or another) SWCV of same repository at end of development</td>
<td>Within one Repository: SWCV1-&gt;SWCV2, single objects or all objects of a namespace or all objects of a SWCV</td>
<td>Identical object with same version as source object</td>
</tr>
<tr>
<td>(menu “Tools”)</td>
<td></td>
<td>------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>(Only Repository)</td>
<td></td>
<td>------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Export/Import</td>
<td>Transport, shipment</td>
<td>Repository1 -&gt; Repository2, (all objects of a namespace or all objects of a SWCV), Directory1 -&gt; Directory2</td>
<td>Repository.: Copy with same version as source object</td>
</tr>
<tr>
<td>(context menu)</td>
<td></td>
<td>------------------------------------------------------------------------------------------------------</td>
<td>Directory: New initial object version is created</td>
</tr>
</tbody>
</table>

**SWCV: Software Component Version**
Overview Integration Builder

Mapping – Introduction

- Overview
- Java Mapping
- XSLT Mapping
- ABAP Mapping

Message Mapping
Overview - General Mapping Types

Mapping

- Transformation from one message structure to another
- Transformation rules defined by *mapping program*

```
<hh:book-flight xmlns:hh="...">
  <hh:customer>
    <hh:first-name>Micheal</hh:first-name>
    <hh:surname>Moore</hh:surname>
  </hh:customer>
  <hh:booking-data>
    <hh:airline>Airline Kingdom</hh:airline>
    <hh:connection>AK087</hh:connection>
    <hh:date>2001-11-22</hh:date>
    <hh:class>business</hh:class>
  </hh:booking-data>
</hh:book-flight>
```

```
<ak:BookFlight xmlns:ak="...">
  <ak:Date>22.11.2001</ak:Date>
  <ak:Flight>087</ak:Flight>
  <ak:Passenger>Micheal Moore</ak:Passenger>
  <ak:Class>2</ak:Class>
  <ak:Agency>Happy Holiday</ak:Agency>
</ak:BookFlight>
```
Messages in XI contain a sender interface
IS evaluates configuration data of the Integration Directory
• During logical routing, the IS determines a Receiver Interface (1)
• Such an interface pair can optionally reference an Interface Mapping
• An interface mapping points to one or more mapping programs (for request, response and/or fault messages)
• IS executes mapping program (2)
• Receiver determination and mapping step can also be performed within the scope of BPM (1;2)
Overview - Relate Mapping Programs to Interfaces

Integration Repository

<table>
<thead>
<tr>
<th>Outbound Message Interface</th>
<th>Interface Mapping(s)</th>
<th>Inbound Message Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Request</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output Message Type</td>
<td>Message Mapping</td>
<td>Input Message Type</td>
</tr>
<tr>
<td></td>
<td>XSLT Mapping</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Java Mapping</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABAP Mapping</td>
<td></td>
</tr>
<tr>
<td><strong>Response</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input Message Type</td>
<td>Message Mapping</td>
<td>Output Message Type</td>
</tr>
<tr>
<td></td>
<td>XSLT Mapping</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Java Mapping</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABAP Mapping</td>
<td></td>
</tr>
<tr>
<td><strong>Fault</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fault Message Type</td>
<td>Message Mapping</td>
<td>Fault Message Type</td>
</tr>
<tr>
<td></td>
<td>XSLT Mapping</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Java Mapping</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABAP Mapping</td>
<td></td>
</tr>
</tbody>
</table>
Overview - Mapping Programs to Map Messages

Message Mapping
- Designed by using the graphical mapping editor of Integration Builder
- Result: Generated Java Code.

Imported Archives
- Import externally defined mapping programs into repository
- Java mapping:
  Implemented by using a specific interface
- XSLT mapping:
  - Runtime supports XSLT processor
  - Java methods can be called from within a Style Sheet

ABAP Mapping
You can execute mapping programs in a sequence
(Only for a request or a response message)
Overview Integration Builder

Mapping – Introduction

- Overview
- Java Mapping
- XSLT Mapping
- ABAP Mapping

Message Mapping
Mapping programs can be implemented in Java

1. Develop your Java code locally e.g. with your IDE
2. Create a .jar or a .zip
   2.1 Using the jar command of the jdk
   2.2 Using a built-in IDE functionality
3. Create a new **Imported Archive** and import the .jar to the Integration Repository
Define a Java class that implements the Java interface com.sap.aii.mapping.api.StreamTransformation.

The interface contains two methods:

1. `public void execute(java.io.InputStream in, java.io.OutputStream out)`
   to perform the mappings.
2. `public void setParameter(java.util.Map param)`
   to access to runtime constants possible (for example: INTERFACE, SENDER_NAME)

The required libraries are contained in the aii_map_api.jar, which is part of your SAP J2EE Engine installation.

DOM and SAX Parsers provided by the SAP XML Toolkit for Java can be used.

API to execute value mappings.
```java
import com.sap.iai.mapping.api.StreamTransformation;
import com.sap.iai.mapping.api.StreamTransformationConstants;

public class JavaMapping implements StreamTransformation {
    private Map myParam;

    public void setParameter(Map param) {
        myParam = param;
    }

    public void execute(InputStream in, OutputStream out) {
        // ...
        String senderName = (String)myParam.get(
            StreamTransformationConstants.SENDER_NAME);
        // ...
    }
}
```
Overview Integration Builder

Mapping – Introduction

- Overview
- Java Mapping
- XSLT Mapping
- ABAP Mapping

Message Mapping
XSLT Mapping (I)

Proceeding equivalent as for Java mappings
1. Develop your XSLT files
2. Create a .jar or a .zip
   2.1 Using the jar command of your jdk
   2.2 Using a built-in IDE functionality
3. Create a new Imported Archive and import the .jar to the Integration Repository
4. Uploaded XSLT programs can modified within the XI Repository by means of a simple editor.
XSLT Mapping (II) - Features

- XSL-Transformation: transforming XML structures
- XPath: Locating information in XML documents
- `<xsl:include>` and `<xsl:import>`: inclusion of other XSLT files
- `<xsl:param>` to access runtime constants
- Invoking external Java methods from within an XSL Transformation
XSLT Mapping with Java Enhancement

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:javamap="java:com.company.group.MappingClass">
  <xsl:param name="first" select="/first-name"/>
  <xsl:param name="last" select="/last-name"/>
  <xsl:template match="/">
    ...
    <name>
      <xsl:if test="function-available('javamap:merge')">
        <xsl:value-of select="javamap:merge($first, $last)"/>
      </xsl:if>
      </name>
    ...
  </xsl:template>
</xsl:stylesheet>
```

Calls the method `merge`:

```java
package com.company.group;

public class MappingClass {
  ...
  public static String merge(String first, String last) {
    return first + ' ' + last;
  }
  ...
}
```
Overview Integration Builder

Mapping – Introduction

- Overview
- Java Mapping
- XSLT Mapping
- ABAP Mapping

Message Mapping
Overview Integration Builder

Mapping – Introduction

Message Mapping

- General Concepts
- Standard Functions
- Element Cardinality
- Context Handling
- User-Defined Functions
- Value Mapping
- Special Features
Overview Integration Builder

Mapping – Introduction

Message Mapping

- General Concepts
- Standard Functions
- Element Cardinality
- Context Handling
- User-Defined Functions
- Value Mapping
- Special Features
General Concepts - Mapping Editor (I)

Data-Flow Editor

Structure Overview

Source Structure

Target Structure

Target Field Mapping
General Concepts - Mapping Editor (II)

Import of source and target structure

- From Integration Repository
  - XSDs developed in the Integration Repository
  - Imported External Definitions (WSDLs, XSDs, DTDs)
- From local file system (XSD or XML)
  - Not all XSD tags are supported (general rule: XSDs that can be created by using the data type editor can be imported)
  - `<xsd:include>` and `<xsd:import>` are not supported
- Mapping editor displays a simplified XML representation of XSD files in structure overview
Design message mapping using the data-flow editor

- Message mapping is composed of several *target field mappings*
- You can edit one target field mapping in the data-flow editor
- All mandatory target fields (marked in red) must be mapped
General Concepts - Elements

- Attribute

- Element is mandatory and might re-occur ($min\text{Occurs}>0$, $max\text{Occurs}>1$)

- Element is mandatory ($min\text{Occurs}>0$)

- Parent node might repeat ($max\text{Occurs}>1$)

- Element might repeat ($max\text{Occurs}>1$)
General Concepts - Elements

Elements

Nodes
- Do not contain any values

Attributes
- Attributes behave like elements in mappings
- Attributes are marked with “@” sign
General Concepts – Assignment of Fields

**Double click on an any field**
- insert field into data-flow editor

**Double click on a “white” field**
- create a new mapping

**Double click on a green/yellow field**
- navigate to existing mapping

**drag & drop**
- New field added to data-flow editor

**drag & drop**
- Field moved within data-flow editor

**Double click on a green/yellow field**
- Navigate to existing mapping
1. Mark root nodes of subtrees

2. Choose assignment function

- Mappings are created automatically for homonymous child elements
General Concepts – Mapping Templates

1. Mark root nodes of subtrees
2. Choose “Load Mapping Template”

- Mapping Templates based on Date Types have to be created beforehand
- Subnodes must reference this pair of Data Types
### General Concepts - Text Preview

#### All target field mappings

<table>
<thead>
<tr>
<th>Contact</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>type</td>
</tr>
<tr>
<td>Occurrences</td>
<td>Occurrences</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
</tr>
<tr>
<td>Det</td>
<td>Det</td>
</tr>
</tbody>
</table>

```
In Contact: ContactID = ins:ContactRefNo
In Contact: ContactName = concat (ins:ContactNameTitle, ins:ContactNameFirstname, ins:ContactNameSurname)
In Customer: CustomerID = ins:CustomerID
In Customer: CustomerName = ins:CustomerID
In Customer: CustomerName = ins:CustomerNameSurname
In Customer: CustomerAddress = ins:CustomerAddress
In Customer: CustomerAddress = ins:CustomerAddressStreet
In Customer: CustomerAddress = ins:CustomerAddressZIPCode
In Customer: CustomerAddress = ins:CustomerAddressCity
```
General Concepts - Dependencies

All mappings or mappings of two selected subnodes are displayed.
1. Type in test parameters in template.

2. Execute test mapping
General Concepts – Test Mappings (II)

1. Upload xml instances from local file system…
   ... or type xml instance directly.

2. Execute test mapping
General Concepts – Multi-Mappings

- Used within Business Process Management
- Development alike to Message Mappings
- N sender messages can be mapped to M receiver messages
- Messages may repeat
- Used to map abstract interfaces
- Original data types wrapped by 
  <Messages>  
  <MessagesN> tags

Source Message

Target Message

Mapping-Programm

© SAP AG 2002, Title of Presentation, Speaker Name
Overview Integration Builder

Mapping – Introduction

Message Mapping

- General Concepts
- Standard Functions
- Element Cardinality
- Node Functions
- Value Mapping
- User-Defined Functions
- Special Features
Standard Functions: Introduction

- All functions expect and return string values
- Generating functions ‘produce’ values (for example: Constant)
- Some standard functions provide function properties (denoted by a small asterisk)
  - For example: `concat` lets you additionally specify a delimiter
  - Call the function properties window by double clicking the function
Simple Standard Functions: \texttt{concat}

Function category: \textit{Text}

<table>
<thead>
<tr>
<th>Source Message</th>
<th>Mapping</th>
<th>Target Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;}&lt;Customer&gt;&lt;Firstname&gt;Harry&lt;/Firstname&gt;&lt;Surname&gt;Potter&lt;/Surname&gt;&lt;/Customer&gt;</td>
<td>\texttt{concat properties Delimiter=&quot; &quot;}</td>
<td>\texttt{&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;}&lt;Contact&gt;&lt;Fullname&gt;Harry Potter&lt;/Fullname&gt;&lt;/Contact&gt;</td>
</tr>
</tbody>
</table>
**Simple Standard Functions: Substring**

**Function category: Text**

<table>
<thead>
<tr>
<th>Source Message</th>
<th>Mapping</th>
<th>Target Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code>&lt;Contact&gt;&lt;RefNo&gt;0123456789-181170&lt;/RefNo&gt;&lt;/Contact&gt;`</td>
<td><img src="image" alt="Diagram" /></td>
<td><code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code>&lt;Customer&gt;&lt;CustomerID&gt;0123456789&lt;/CustomerID&gt;&lt;/Customer&gt;`</td>
</tr>
</tbody>
</table>

**Substring properties**
- start from: 0
- count: 10

- counting starts at 0!
## Simple Standard Functions: DateTrans

### Function category: Date

<table>
<thead>
<tr>
<th>Source Message</th>
<th>Mapping</th>
<th>Target Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</td>
<td></td>
<td>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</td>
</tr>
<tr>
<td>&lt;Customer&gt; &lt;custBirthday&gt; 181170 &lt;&lt;/custBirthday&gt; &gt;</td>
<td></td>
<td>&lt;Contact&gt; &lt;birthday&gt; 11/18/1970 &lt;/birthday&gt; &gt;</td>
</tr>
<tr>
<td></td>
<td>custBirthday</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DateTrans</td>
<td>birthday</td>
</tr>
</tbody>
</table>

**DateTrans properties**

src Format: ddMMyy  
dst Format: MM/dd/yyyy

For potential date patterns, please see Java class `java.text.DateFormat`
**Boolean Functions**

- Return `true` or `false` string
- Interpret `1` and `true` (not case-sensitive) as `true-value`
### Calculations

**Function category: Arithmetic**

<table>
<thead>
<tr>
<th>Source Message</th>
<th>Mapping</th>
<th>Target Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code> <code>&lt;Customer&gt;</code> <code>&lt;debt&gt;</code> 90 <code>&lt;/debt&gt;</code> &lt;/Customer&gt;`</td>
<td><img src="image" alt="Diagram" /></td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code> <code>&lt;Contact&gt;</code> <code>&lt;debt&gt;</code> 90.00 <code>&lt;/debt&gt;</code> &lt;/Contact&gt;`</td>
</tr>
</tbody>
</table>

**FormatNum properties**

Number Format: `###,####,####.00`

(For potential patterns, please see Java class `java.text.DecimalFormat`)

- Calculations (add, multiply, etc.) are performed with Java float precision
- Trailing zeros in result are truncated, e.g.: `4.2-0.2 = 4`
- To keep trailing zeros use `FormatNum` function
Overview Integration Builder

Mapping – Introduction

Message Mapping

- General Concepts
- Standard Functions
- Element Cardinality
- Node Functions
- Value Mapping
- User-Defined Functions
- Special Features
Element Cardinality – Mandatory Mappings 
*(minOccurs>0)*

Mandatory Elements and Nodes 
*(minOccurs>0)*

- Mapping is mandatory. A source element or a constant has to be assigned to the target element.
- Otherwise error message: “Message-mapping not completely defined”
Exception: Nodes with fixed occurrences ($minOccurs=maxOccurs$)

- Nodes are automatically marked in green.
- Mapping is not mandatory. 
  $<minOccurs>$ Nodes are created automatically.

For elements with fixed occurrences
the rules of mandatory elements are applied
Element Cardinality – Repeating Elements
(maxOccurs>1)

Mapping of Repeating Elements and Nodes (maxOccurs>1)

■ Source: Target element is created as often as mapped source element occurs in the source message.

■ Source: Creating function (e.g. Constant)
  ◆ If minOccurs>0, <minOccurs> target elements are created.
  ◆ If minOccurs=0, 1 target element is created.
Mapping of non mandatory Nodes (*minOccurs*=0)

If a non mandatory node contains elements with *minOccurs*>0, these elements become mandatory after their parent node is assigned.
Mapping of non mandatory Nodes (*minOccurs*=0)

- Assignment of elements is not sufficient
- Node has to be ‘created’ by assigning a suitable source node/element
Element Cardinality – Unique Target Mapping

Mapping from more than one Source element

- Only a single and unique mapping relation per target element is allowed (n:1 relationship)

- Multiple mapping relations can be defined by duplicating the target node or element (Duplicate Subtree)

- To duplicate a node or an element, right-click on the object in the target structure pane and select “Duplicate Subtree”
Overview Integration Builder

Mapping – Introduction

Message Mapping

- General Concepts
- Standard Functions
- Element Cardinality
- Context Handling
- Value Mapping
- User-Defined Functions
- Special Features
Context Change

- Message mapping works internally by using queues
- If no further elements are imported at a particular hierarchy level, a *Context Change* is inserted in the queue
- Use node functions to handle changes in the message hierarchy.

```
<root>
    <item> 3 </item>
    <item> 2 </item>
</root>
```

```
<root>
    <item> 10 </item>
    <item> 11 </item>
    <item> 43 </item>
    <item> 2 </item>
</root>
```

```
<root>
    <item> 7 </item>
</root>
```
Context Handling (II)

- **Why Queue Processing?**
  - No size limitations for messages
  - Better runtime performance

- **Context changes have impact on:**
  - User-Defined Functions
  - Breaking and inserting of hierarchy levels

- **Manipulation of queues and contexts**
  - Explicit context selection on source elements and nodes
  - Using node functions
    - `removeContexts`: deletes all context changes of a queue
    - `SplitByValue`: insert additional context changes in a queue
A tool to display queues:

1. Upload or create source XML document in test mode
2. Right-click on box representing element or function
3. Select “Show queue”
Context Handling - “Show queue” (II) Example

Sender and Receiver Structure

<table>
<thead>
<tr>
<th>LANGUAGE</th>
<th>TEXT</th>
<th>LANGUAGE</th>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATNO 4711</td>
<td>Zahnpasta</td>
<td>LANGKEY EN</td>
<td></td>
</tr>
<tr>
<td>MATTEXT toothpaste</td>
<td>LANGKEY FR</td>
<td>TEXT dentifrice</td>
<td></td>
</tr>
</tbody>
</table>

Related Mapping

- LANGKEY
- EN
- TEXT
- Description
“Show queue” function display value queues leaving the related element/function.

“Debugging” functionality

Applicable to any step in the mapping

Step by step check of mappings
Context Handling - “Show queue” (IV)

Initial and terminal context change – Not passed to the internal queue processing

The displayed queue contains the following values:

- ResultList.SUPPRESS
- ResultList.CC
- “toothpaste”
- ResultList.CC
- ResultList.SUPPRESS
Context Handling - Node Functions – removeContexts(I)

With `removeContexts`

<table>
<thead>
<tr>
<th>Source Message</th>
<th>Mapping</th>
<th>Target Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code></td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code></td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Test_Out_Remove&gt;</code></td>
<td><code>&lt;header name=&quot;A&quot;&gt;</code></td>
<td><code>&lt;header name=&quot;B&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;header name=&quot;A&quot;&gt;</code></td>
<td><code>&lt;item&gt;A.one&lt;/item&gt;</code></td>
<td><code>&lt;item&gt;B.one&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;item&gt;A.two&lt;/item&gt;</code></td>
<td><code>&lt;item&gt;A.three&lt;/item&gt;</code></td>
<td><code>&lt;item&gt;B.two&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;header&gt; &lt;/header&gt;</code></td>
<td><code>&lt;header&gt; &lt;/header&gt;</code></td>
<td><code>&lt;header&gt; &lt;/header&gt;</code></td>
</tr>
<tr>
<td><code>&lt;header name=&quot;B&quot;&gt;</code></td>
<td><code>&lt;item&gt;B.one&lt;/item&gt;</code></td>
<td><code>&lt;item&gt;B.three&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;item&gt;B.two&lt;/item&gt;</code></td>
<td><code>&lt;item&gt;B.three&lt;/item&gt;</code></td>
<td><code>&lt;item&gt;B.three&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;header&gt; &lt;/header&gt;</code></td>
<td><code>&lt;header&gt; &lt;/header&gt;</code></td>
<td><code>&lt;header&gt; &lt;/header&gt;</code></td>
</tr>
</tbody>
</table>

- **use `removeContext()` to delete the parent context of an element.**
## Without removeContexts

<table>
<thead>
<tr>
<th>Source Message</th>
<th>Mapping</th>
<th>Target Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code></td>
<td></td>
<td><code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Test_Out_Remove&gt;</code></td>
<td></td>
<td><code>&lt;Test_In_Remove&gt;</code></td>
</tr>
<tr>
<td><code>&lt;header name=&quot;A&quot;&gt;</code></td>
<td></td>
<td><code>&lt;item&gt;A.one&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;item&gt;A.one&lt;/item&gt;</code></td>
<td></td>
<td><code>&lt;item&gt;A.two&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;item&gt;A.two&lt;/item&gt;</code></td>
<td></td>
<td><code>&lt;item&gt;A.three&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;item&gt;A.three&lt;/item&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;header name=&quot;B&quot;&gt;</code></td>
<td></td>
<td><code>&lt;item&gt;B.one&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;item&gt;B.one&lt;/item&gt;</code></td>
<td></td>
<td><code>&lt;item&gt;B.two&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;item&gt;B.two&lt;/item&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;header&gt;</code></td>
<td></td>
<td><code>&lt;item&gt;A.three&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Test_Out_Remove&gt;</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**

```
  item 1  item 2
  ↑      ↓
```

© SAP AG 2002, Title of Presentation, Speaker Name 61
With `SplitByValue`

<table>
<thead>
<tr>
<th>Source Message</th>
<th>Mapping</th>
<th>Target Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code></td>
<td></td>
<td><code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; ?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Test_Out_Split&gt;</code></td>
<td></td>
<td><code>&lt;Test_In_Split&gt;</code></td>
</tr>
<tr>
<td><code>&lt;header&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;item&gt;one&lt;/item&gt;</code></td>
<td></td>
<td><code>&lt;new_context&gt;</code></td>
</tr>
<tr>
<td><code>&lt;item&gt;two&lt;/item&gt;</code></td>
<td><code>item</code></td>
<td><code>&lt;item&gt;one&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;item&gt;three&lt;/item&gt;</code></td>
<td><code>new_context</code></td>
<td><code>item&gt;one&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;item&gt;four&lt;/item&gt;</code></td>
<td></td>
<td><code>&lt;new_context&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/header&gt;</code></td>
<td></td>
<td><code>&lt;item&gt;two&lt;/item&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/Test_Out_Split&gt;</code></td>
<td></td>
<td><code>&lt;new_context&gt;</code></td>
</tr>
</tbody>
</table>

- `SplitByValue()` is the counterpart to `removeContexts()`
- inserts a context change in the source value queue.
- context change in the queue after each value, after each change to the value, or after an empty tag.
### Without SplitByValue

<table>
<thead>
<tr>
<th>Source Message</th>
<th>Mapping</th>
<th>Target Message</th>
</tr>
</thead>
</table>
| `<?xml version="1.0" encoding="UTF-8" ?>
<Test_Out_Split>
<header>
  <item>one</item>
  <item>two</item>
  <item>three</item>
  <item>four</item>
</header>
</Test_Out_Split>` | ![Mapping Diagram] | `<?xml version="1.0" encoding="UTF-8" ?>
<Test_In_Split>
  <new_context>
    <item>one</item>
    <item>two</item>
    <item>three</item>
    <item>four</item>
  </new_context>` |
Context Handling – Explicit Context Selection (I)

- As of XI 2.0 SP3 the context of source nodes and elements can be selected *explicitly*
- More precise handling as `removeContexts` function
- **But**: `removeContexts` function still necessary for context manipulation within a mapping chain
- Example:

<table>
<thead>
<tr>
<th>Message Type</th>
<th>XML representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test_Context_Out</td>
<td>Test_Context_O...1..1</td>
</tr>
<tr>
<td>parent</td>
<td></td>
</tr>
<tr>
<td>segment</td>
<td></td>
</tr>
<tr>
<td>item</td>
<td>A.1.1</td>
</tr>
<tr>
<td>item</td>
<td>A.1.2</td>
</tr>
<tr>
<td>segment</td>
<td></td>
</tr>
<tr>
<td>item</td>
<td>A.2.1</td>
</tr>
<tr>
<td>segment</td>
<td></td>
</tr>
<tr>
<td>item</td>
<td>A.3.1</td>
</tr>
<tr>
<td>parent</td>
<td></td>
</tr>
<tr>
<td>segment</td>
<td></td>
</tr>
<tr>
<td>item</td>
<td>B.1.1</td>
</tr>
<tr>
<td>segment</td>
<td></td>
</tr>
<tr>
<td>item</td>
<td>B.2.1</td>
</tr>
<tr>
<td>item</td>
<td>B.2.2</td>
</tr>
</tbody>
</table>
Context Handling – Explicit Context Selection (II)

- Explicit choice of contexts and resulting queues
- Context changes are relevant:
  - user-defined functions
  - structure hierarchy
- **But:** the `removeContexts` function is still necessary for context manipulation within a mapping chain:

<table>
<thead>
<tr>
<th>Context</th>
<th>Resulting Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Context 1" /></td>
<td><img src="image2" alt="Queue 1" /></td>
</tr>
<tr>
<td><img src="image3" alt="Context 2" /></td>
<td><img src="image4" alt="Queue 2" /></td>
</tr>
<tr>
<td><img src="image5" alt="Context 3" /></td>
<td><img src="image6" alt="Queue 3" /></td>
</tr>
</tbody>
</table>
Context Handling – Explicit Context Selection (I)

- Comparing two contexts with diverse numbers of entries:
  - Last value of context with less elements is reiterated
  - Applies also to Constants

Example: the content of the two contexts is concatenated; Value “prodAttribute” of /Fieldnames is reiterated to fill up result context.
Overview Integration Builder

Mapping – Introduction

Message Mapping

- General Concepts
- Standard Functions
- Element Cardinality
- Context Handling
- Value Mapping
- User-Defined Functions
- Special Features
Value Mapping (I) – FixValues

The FixValues function provides a simple generic value mapping (or key mapping) in the Integration Repository.

1. Use the standard function Conversions -> FixValues to link source and target element.

2. Double click to define properties:
   - Each field mapping uses its own table of key-value pairs.
   - Filled tables can be copied to other mappings. But copied table is independent of origin.
The Value mapping function provides a more specific value transformation.

1. Use the standard function Conversions -> Value mapping to link source and target element
2. Double click to define properties
3. Define access keys for table entries:
   - Value Mapping Context stores information on the origin of the data. If data is maintained in the Integration Directory, use http://sap.com/xi/XI
   - Agency and Schema for sender and receiver messages. Any arbitrary string value can be used.
Semantically identical information is collected in “Value Mapping Groups”

Value Mapping Groups can be maintained:
- in the Integration Directory
- by a Java or ABAP proxies built upon Interface “ValueMappingReplication” of SAP BASIS -> SAP BASIS 6.40 -> http://sap.com/xi/XI/System

Following entries are required
1. Agency
2. Schema (Namespace and Object Type )
3. Value (the group’s representative for a given Agency – Schema combination)
Value Mappings are created automatically between representatives of the same group.

All entries for a couple of Agency/Schema combinations are displayed using *Tools-*->* Value Mapping* in the Integration Directory.

Each new entry will create automatically a new group.
Value Mapping (VI) – Mass Configuration

- Mass data can be maintained in Value Mappings tables by means of an XI Interface
- Based on Message Interface
  http://sap.com/xi/XI/System
  ValueMappingReplication of SC SAP BASIS 6.40
- Configuration has to be created in Directory
- Provided data has to correspond to structure:
Overview Integration Builder

Mapping – Introduction

Message Mapping

- General Concepts
- Standard Functions
- Element Cardinality
- Node Functions
- Value Mapping
- User-Defined Functions
- Special Features
User-Defined Functions – Overview

- Functional enhancements if standard functions do not fulfill requirements
- Integration of Java programs from imported archives of the same namespace and software component version
- Usage just like standard functions
- Two types: “Simple Functions”, “Advanced Functions”
User-Defined Functions – Definition

Specify:

- **Label:**
  Text that appears in the function chooser and on the data-flow object to identify the new function.

- **Argument Count:**
  The number of input values that should be transferred to the function. Input values are of type `java.lang.String`.

- **Description:**
  Description of the function’s use.
Simple editor functions

Usage of Java programs from imported archives of the same software component version

```java
public String changeValue(String a) {
    // write your code here
    if (a.equals("Business")) {
        return "E";
    }
    else if (a.equals("Private")) {
        return "P";
    }
    else
        return "O";
}
```
Example: following class should enhance a user-defined function

```java
package myPackage;

import com.sap.aii.mappingtool.tf.rt.Container;

public class myClass {

    public static String addCounter(String a, Container container) {

        Object o = container.getParameter("counter");
        Integer ii;
        if (o == null) ii = new Integer(0);
        else ii = (Integer) o;
        ii = new Integer(ii.intValue() + 1);
        container.setParameter("counter", ii);
        return ii.toString();
    }

}
```
User-Defined Functions – Binding of external Classes (II)

Procedure:

1. Compile your Java class and create a .jar
2. Create a new Imported Archive and upload your .jar.
3. The relevant class has to be declared in the Import statement
## User-defined functions – function types

<table>
<thead>
<tr>
<th></th>
<th>Simple Functions</th>
<th>Advanced Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Parameters</strong></td>
<td>String, …, Container</td>
<td>String[], …, ResultList, Container</td>
</tr>
<tr>
<td><strong>Returning Values</strong></td>
<td>return String;</td>
<td>result.addValue(String);</td>
</tr>
<tr>
<td><strong>Processing</strong></td>
<td>Function is called once per field mapping</td>
<td>Function is called once per context or queue. Values of context /queue are stored in input string arrays</td>
</tr>
<tr>
<td></td>
<td>com.sap.aii.mappingtool.tf rt (aii_mt_rt.jar)</td>
<td></td>
</tr>
</tbody>
</table>
Input parameters: `String[]`, `ResultList`, `Container`  

Flag “Cache the whole queue”
  - Selected: all values of the whole queue are passed to the function
  - Not selected: values of one context is passed to the function

`ResultList` contains output of function  
Output builds up a new queue
### User-defined functions – String[]

- Context changes are treated as fields
- They are represented by class variable `java.lang.String ResultList.CC`
- Queue output determines input values of user-defined functions
- First and last context change of each queue are suppressed

<table>
<thead>
<tr>
<th>Caching the whole queue</th>
<th>Not caching the whole queue</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String[] a =</code>&lt;br&gt;<code>{&quot;A.1.1&quot;,&quot;A.1.2&quot;, ResultList.CC,&quot;A.2.1&quot;, ResultList.CC,&quot;A.3.1&quot;, ResultList.CC,&quot;B.1.1&quot;, ResultList.CC,&quot;B.2.1&quot;, &quot;B.2.2&quot;};</code></td>
<td><code>String[] a =</code>&lt;br&gt;<code>{&quot;A.1.1&quot;, &quot;A.1.2&quot;};</code>&lt;br&gt;<code>{&quot;A.2.1&quot;};</code>&lt;br&gt;<code>{&quot;A.3.1&quot;};</code>&lt;br&gt;<code>{&quot;B.1.1&quot;};</code>&lt;br&gt;<code>{&quot;B.2.1&quot;,&quot;B.2.2&quot;};</code></td>
</tr>
</tbody>
</table>
| `String[] a =`<br>`{"A.1.1", "A.1.2"};`<br>`{"A.2.1"};`<br>`{"A.3.1"};`<br>`{"B.1.1"};`<br>`{"B.2.1","B.2.2"};` | }
User-defined functions – Container

- **Container supports the methods**
  - void setParameter(String, Object)
  - Object getParameter(String)
  - Stalled parameters can be accessed in this specific function
    - GlobalContainer getGlobalContainer()
    - MappingTrace getTrace()
    - Map getTransformationParameters()
  - Map can be used to get runtime constants, e.g.
    - Object map.get(StreamTransformationConstants.SENDER_SYSTEM)

- **GlobalContainer supports the methods**
  - void setParameter(String, Object)
  - Object getParameter(String)
  - Stalled parameters can be accessed in different functions of one message mapping

- **MappingTrace supports the methods**
  - void addWarning(String) – trace level: 1, 2, 3
  - void addInfo(String) – trace level: 2, 3
  - void addDebugMessage(String) – trace level: 3
  - Writes String to message monitoring
User-defined functions – ResultList

- ResultList supports the methods and constants
  - void addValue(String)
  - void addContextChange()
  - void addSuppress()
  - void clear()
  - ResultList.CC (context change)
  - ResultList.SUPPRESS (suppressed value)

- Outbound queue has to be build up manually, i.e. in case you cache the whole queue values, context changes and suppressed values have to be added manually.
Overview Integration Builder

Mapping – Introduction

Message Mapping

- General Concepts
- Standard Functions
- Element Cardinality
- Node Functions
- Value Mapping
- User-Defined Functions
- Examples
Examples – Get value by key (I)

Problem:
- A specific value should be filtered by a key or qualifier
- Example: only English <TEXT> should be mapped to resulting field <Description>

```xml
<?xml version="1.0" encoding="UTF-8" ?>
- <MATERIAL>
  - <MATNO>4711</MATNO>
  - <MATTTEXT>
    - <LANGKEY>DE</LANGKEY>
    - <TEXT>Zahnpasta</TEXT>
  </MATTTEXT>
  - <MATTTEXT>
    - <LANGKEY>EN</LANGKEY>
    - <TEXT>toothpaste</TEXT>
  </MATTTEXT>
  - <MATTTEXT>
    - <LANGKEY>FR</LANGKEY>
    - <TEXT>dentifrice</TEXT>
  </MATTTEXT>
</MATERIAL>
```

<table>
<thead>
<tr>
<th></th>
<th>Occurences</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductDefinition</td>
<td>1..1</td>
</tr>
<tr>
<td>RefNo</td>
<td>1..1</td>
</tr>
<tr>
<td>Description</td>
<td>1..1</td>
</tr>
</tbody>
</table>
Solution:

1. Create mapping

2. Set Contexts of `<LANGKEY>` and `<TEXT>`
### Discussion:

<table>
<thead>
<tr>
<th>3. Display Queues to investigate</th>
<th><img src="https://via.placeholder.com/150" alt="Diagram" /></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>LANGKEY</strong> = <strong>EN</strong></td>
</tr>
<tr>
<td></td>
<td><strong>TEXT</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. Explanation</th>
<th>Since the element <code>&lt;Description&gt;</code> may only occur once in the resulting structure, the output has to be put in a single context. The two suppressed values are removed automatically when the resulting document is created.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In an alternative solution two contexts of <code>&lt;LANGKEY&gt;</code> and <code>&lt;TEXT&gt;</code> keep their initial contexts and a <code>removeContexts</code> is inserted after the <code>ifWithoutElse</code> function.</td>
</tr>
</tbody>
</table>
Problem:
- A hierarchical structure has to be mapped to a flat structure
- One <row> should be created per <Item>
- Header information (represented <Header>) occurring only once per <Order> should be multiplied and copied to <row>.
Examples – Copy head to line (II)

Solution:

1. Create User-Defined Function

```java
public void copyPerValue(String[] a, String[] b, ResultList result, Container container)
{
    //write your code here
    for (int i=0; i<a.length; i++) {
        result.addValue(b[i]);
    }
}
```

2. Create mappings

```
Item | row
-----|-----
```

```
Item | copyPerValue | SplitByValue | address
-----|-----|-----|-----
```

© SAP AG 2002, Title of Presentation, Speaker Name
### Discussion:

#### 3. Display Queues

<table>
<thead>
<tr>
<th>Display queue</th>
<th>Display queue</th>
<th>Display queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>PurchaseOrder/Orders...</td>
<td>copyPerValue</td>
<td>SplitByValue</td>
</tr>
<tr>
<td>(Smith)</td>
<td>(Smith)</td>
<td>(Smith)</td>
</tr>
<tr>
<td>(Smith)</td>
<td>(Smith)</td>
<td>(Smith)</td>
</tr>
<tr>
<td>Jones</td>
<td>Jones</td>
<td>Jones</td>
</tr>
<tr>
<td>Item</td>
<td>Item</td>
<td>Item</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
<td>Name</td>
</tr>
</tbody>
</table>

#### 4. Explanation

The User-Defined Function `copyPerValue` copies the value of `<Address>` per each occurrence of `<Item>` to the relevant context. The `SplitByValue` function then distributes the copied values to the contexts of the flat structure.

Since `<Item>` only contains empty fields, you can use in this case also function `concat` instead of `copyPerValue`. See [Context Handling – Explicit Context Selection (I)](#) for details.
Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die ausdrückliche schriftliche Genehmigung durch SAP AG nicht gestattet. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Die von SAP AG oder deren Vertriebsfirmen angebotenen Softwareprodukte können Softwarekomponenten auch anderer Softwarehersteller enthalten.

Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® und SQL Server® sind eingetragene Marken der Microsoft Corporation.

IBM®, DB2®, DB2 Universal Database, OS/2®, Parallel Sysplex®, MVS/ESA, AIX®, S/390®, AS/400®, OS/390®, OS/400®, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere®, Netfinity®, Tivoli®, Informix und Informix® Dynamic ServerTM sind Marken der IBM Corporation in den USA und/oder anderen Ländern.

ORACLE® ist eine eingetragene Marke der ORACLE Corporation.

UNIX®, X/Open®, OSF/1® und Motif® sind eingetragene Marken der Open Group.

Citrix®, das Citrix-Logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® und andere hier erwähnte Namen von Citrix-Produkten sind Marken von Citrix Systems, Inc.


JAVA® ist eine eingetragene Marke der Sun Microsystems, Inc.

JAVASCRIPT® ist eine eingetragene Marke der Sun Microsystems, Inc., verwendet unter der Lizenz der von Netscape entwickelten und implementierten Technologie.

MarketSet und Enterprise Buyer sind gemeinsame Marken von SAP AG und Commerce One.

SAP, SAP Logo, R/2, R/3, mySAP, mySAP.com und weitere im Text erwähnte SAP-Produkte und -Dienst-leistungen sowie die entsprechenden Logos sind Marken oder eingetragene Marken der SAP AG in Deutschland und anderen Ländern weltweit. Alle anderen Namen von Produkten und Dienstleistungen sind Marken der jeweiligen Firmen.